Woche 06: Programmierung – Lineare Regression mit Scikit-learn

Skript

Erarbeitet von
Ludmila Himmelspach

Lernziele

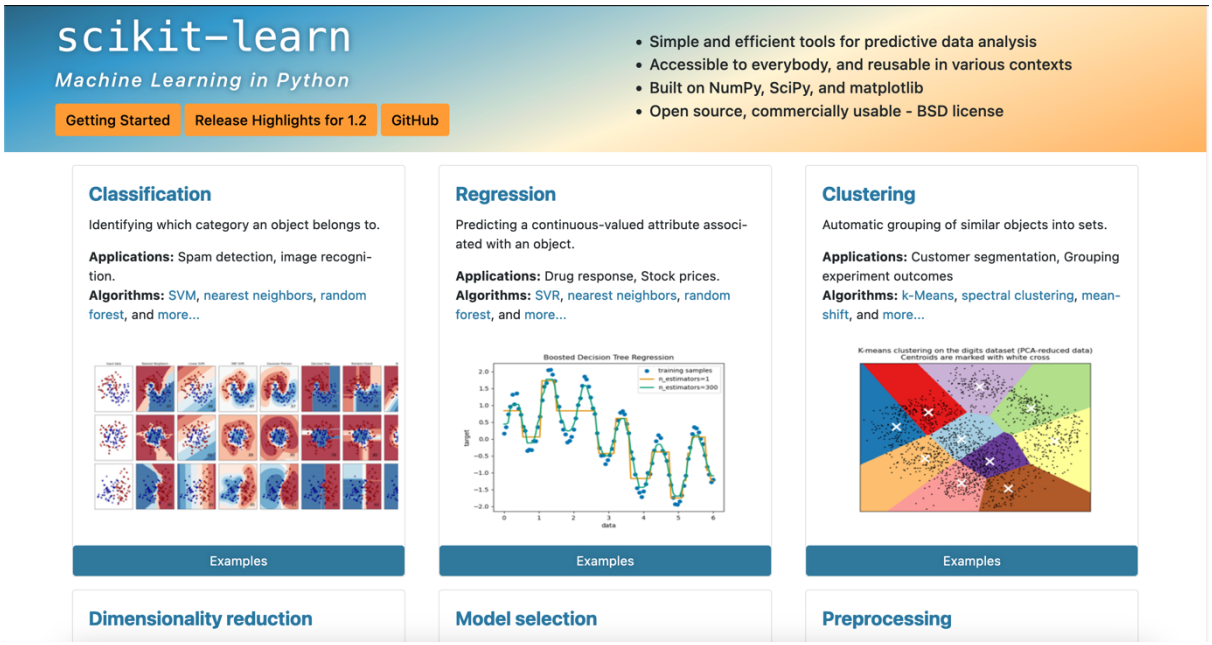
- Referenzdatensätze aus dem Modul *Scikit-learn* in das Programm laden können
- Einen Datensatz mit einer Hilfsfunktion in eine Trainings- und eine Testmenge aufteilen können
- Ein lineares Regressionsmodell für einen gegebenen Datensatz trainieren können
- Ein trainiertes Regressionsmodell nutzen können, um für Beobachtungen der Testmenge Werte für die Zielgröße vorherzusagen

Projekt „KI für alle"
HeiCAD – Heine Center for Artificial Intelligence and Data Science

Inhalt

Einstieg

Bei der Regressionsanalyse geht es darum, einen Zusammenhang zwischen den Werten eines Zielmerkmals – zum Beispiel der Höhe des Hauskredits – und den Werten anderer Merkmale wie Eigenkapital und Einkommen, die auch „erklärende“ oder „beschreibende“ Merkmale genannt werden, herzustellen, um die Werte des Zielmerkmals für neue Beobachtungen voraussagen zu können. Dafür muss ein Modell formuliert werden. Im Python-Modul *Scikit-learn* findest du einige passende Funktionen dazu. *Scikit-learn* ist ein Modul für Machine Learning in Python. Neben Machine Learning Methoden stellt Scikit-learn auch Methoden für die Datenvorverarbeitung und Funktionen zum Laden der Datensätze zur Verfügung, die in der Forschung zum Testen und Vergleichen verschiedener Verfahren eingesetzt werden.



Einblendung Scikit-learn-Webseite (scikit-learn.org)

Quelle [1]

Vorbereitung des Datensatzes

Eine lineare Regression modelliert die Beziehung zwischen den Werten sogenannter „erklärender” Merkmale bzw. Features und den *metrischen* Werten eines Zielmerkmals. Die entsprechende Funktion dafür findest du im Untermodul *Linear Models*. Außerdem benötigen wir noch weitere Untermodule: Das Untermodul *Datasets* enthält

Hilfsprogramme zum Laden von Referenzdatensätzen. Das Untermodul *Model Selection* stellt viele Funktionen zum Aufteilen der Datensätze in *Trainings*- und *Testteilmengen* zur Verfügung. Wir importieren alle diese Untermodule in unser Programm.

```
from sklearn import linear_model, datasets, model_selection
```

In diesem Video arbeiten wir mit dem *California Housing Dataset*, das Informationen über die Blockgruppen in Kalifornien enthält. Dabei entspricht eine Blockgruppe einem Zählungsgebiet, in dem in der Regel zwischen 600 und 3.000 Personen wohnen. Aufgeteilt nach Blockgruppen enthält das California Housing Dataset Informationen, wie das mittlere Einkommen der dort wohnenden Menschen, das mittlere Alter der jeweiligen Häuser und die durchschnittliche Anzahl der Haushaltsmitglieder in einer Blockgruppe. Das Zielmerkmal enthält den mittleren Hauswert (in $100.000) für den kalifornischen Bezirk, in dem sich die Blockgruppe befindet.

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MedHouseVal
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23	4.526
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22	3.585
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24	3.521
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25	3.413
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25	3.422
5	4.0368	52.0	4.761658	1.103627	413.0	2.139896	37.85	-122.25	2.697
6	3.6591	52.0	4.931907	0.951362	1094.0	2.128405	37.84	-122.25	2.992
7	3.1200	52.0	4.797527	1.061824	1157.0	1.788253	37.84	-122.25	2.414
8	2.0804	42.0	4.294118	1.117647	1206.0	2.026891	37.84	-122.26	2.267
9	3.6912	52.0	4.970588	0.990196	1551.0	2.172269	37.84	-122.25	2.611
10	3.2031	52.0	5.477612	1.079602	910.0	2.263682	37.85	-122.26	2.815
11	3.2705	52.0	4.772480	1.024523	1504.0	2.049046	37.85	-122.26	2.418
12	3.0750	52.0	5.322650	1.012821	1098.0	2.346154	37.85	-122.26	2.135
13	2.6736	52.0	4.000000	1.097701	345.0	1.982759	37.84	-122.26	1.913
14	1.9167	52.0	4.262903	1.009677	1212.0	1.954839	37.85	-122.26	1.592
15	2.1250	50.0	4.242424	1.071970	697.0	2.640152	37.85	-122.26	1.400
16	2.7750	52.0	5.939577	1.048338	793.0	2.395770	37.85	-122.27	1.525
17	2.1202	52.0	4.052805	0.966997	648.0	2.138614	37.85	-122.27	1.555

Einblendung Ausschnitt Datentabelle

Quelle [2]

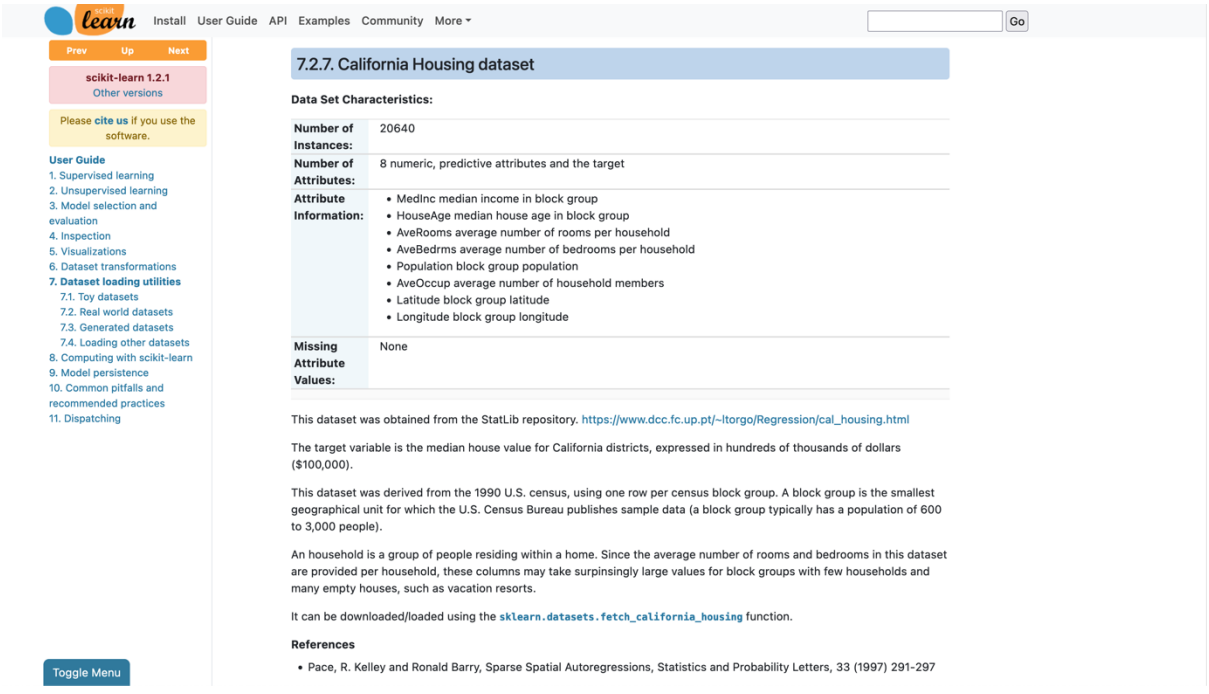
Den Datensatz *California Housing* kannst du mit der Funktion *fetch_california_housing()* in dein Programm laden. Damit das Zielmerkmal von den übrigen Merkmalen unterschieden werden kann, ist es sinnvoll, das Zielmerkmal und die übrigen Merkmale in zwei voneinander getrennte *ndarrays* zu laden. Das erreichst du, indem du den Parameter *return_X_y* der *fetch_california_housing()*–Funktion auf *True* setzt.

Quelle [3]

Wir speichern die Datenmatrix mit den beschreibenden Merkmalen im *ndarray X_california_housing* und die Werte der Zielvariable, also die mittleren Häuserwerte, im *ndarray y_california_housing* ab.

```
# Lade den Datensatz "California Housing"
X_california_housing, y_california_housing =\
    datasets.fetch_california_housing(return_X_y=True)
```

Wenn du dir einige Werte des Datenarrays oder des Zielmerkmals anschauen möchtest, kannst du das entsprechende *ndarrays* mit der *print()*-Funktion ausgeben lassen. Falls du nähere Informationen zu diesem Datensatz und die Beschreibung aller Merkmale wünschst, kannst du diese im User Guide zum *Scikit-learn* finden.



Einblendung User Guide

Quelle [4]

Der Einfachheit halber benutzen wir nur das erste Merkmal, also das mittlere Einkommen in der Blockgruppe, als „erklärendes" Merkmal. Deswegen wählen wir die erste Spalte des Datenarrays *X_california_housing* und speichern das Ergebnis im gleichen Array ab.

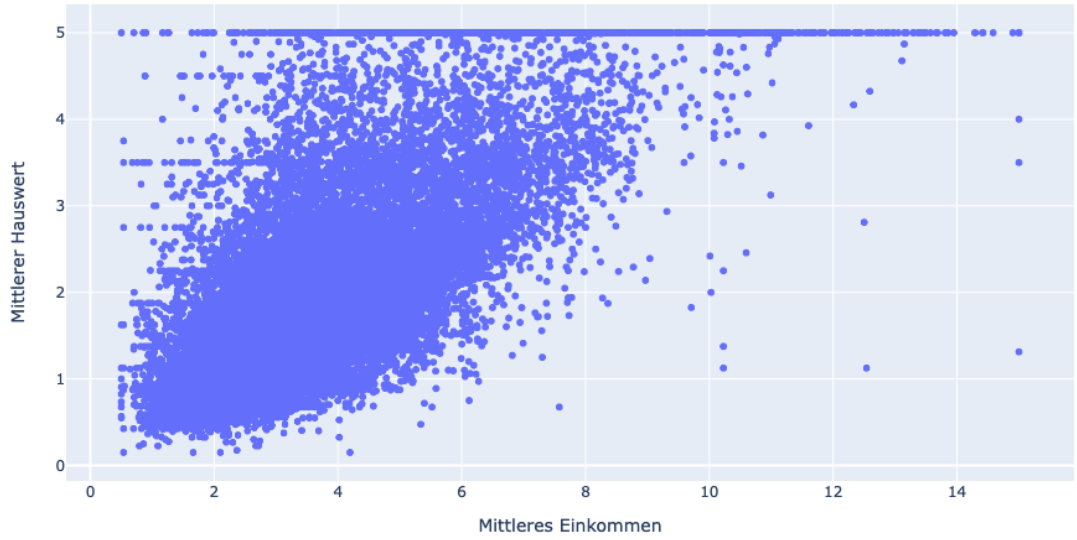
```
# Selektiere nur die erste Spalte des Arrays X_california_housing
X_california_housing = X_california_housing[:, 0]
```

Wir stellen die Beobachtungen durch ihre Werte für das mittlere Einkommen und den mittleren Hauswert grafisch dar, um einen Überblick über die Daten zu bekommen. Dafür importieren wir die benötigten *plotly*-Untermodule. Du kannst die Untermodule auch an dieser Stelle im Notebook in das Programm importieren, aber zum guten Programmierstil gehört es, alle *import*-Anweisungen ganz am Anfang des Programms zu tätigen. Das machen wir jetzt auch.

```
import plotly.express as px
import plotly.offline as pyo
import plotly.graph_objects as go
```

```
pyo.init_notebook_mode()
fig1 = px.scatter(x=X_california_housing, y=y_california_housing)
fig1.update_layout(xaxis_title='Mittleres Einkommen',
                   yaxis_title='Mittlerer Hauswert')
fig1.show()
```

Anhand der Grafik können wir einen linearen Zusammenhang zwischen dem Einkommen und dem Hauswert bereits vermuten, da die Datenwolke tendenziell von unten links nach oben rechts gerichtet ist.



Da wir aus der gesamten Datenmatrix nur ein Merkmal verwendet haben, wurde das Datenarray zu einem eindimensionalen *ndarray* reduziert. Wie wir später sehen werden,

erfordert die Funktion zum Trainieren des Regressionsmodells eine zweidimensionale Datenmatrix als Eingabe. Um das Datenarray um eine Dimension zu erweitern, müssen wir ihm eine zweite Dimension ohne Werte hinzufügen. Dafür verwenden wir den Wert *None*, der in Python die Abwesenheit eines Wertes repräsentiert.

```
# Erweitere das Datenarray um eine Dimension
X_california_housing = X_california_housing[:, None]
print(X_california_housing.shape)
```

In der Ausgabe der *print()*-Anweisung sehen wir, dass unser Datenarray jetzt zweidimensional ist.

Bevor wir ein Regressionsmodell auf den Daten trainieren können, müssen wir die Datenmatrix in eine Trainings- und eine Testmenge zerlegen. Dafür nutzen wir die Funktion *train_test_split()* des Untermoduls von *Scikit-learn Model Selection*.

Quelle [5]

Diese Funktion erwartet als Eingabe die Datenmatrix mit dem entsprechenden Zielmerkmal. Mit dem Parameter *test_size* kannst du die Größe der Testmenge angeben. Wir wollen 90% der Daten zum Trainieren des Modells verwenden, deswegen weisen wir dem Parameter *test_size* den Wert von 0.1 zu. Das bedeutet, dass unsere Testmenge 10 % aller Beobachtungen enthalten wird. In den Voreinstellungen der Funktion ist festgelegt, dass die Beobachtungen vor dem Zerlegen gemischt werden. In manchen Datensätzen sind die Beobachtungen nach dem Zielmerkmal sortiert. Das kann zu einer ungünstigen Aufteilung des Datensatzes in die Trainings- und Testmenge führen, weil die *train_test_split()*-Funktion die ersten, in unserem Fall 90 % der Beobachtungen der Trainingsmenge und die letzten 10 % der Testmenge zuweist. Wenn die Beobachtungen des California Housing Datensatzes nach dem Zielmerkmal Hauswert zum Beispiel aufsteigend sortiert wären, wäre die Trainingsmenge für die gesamte Datenmenge nicht repräsentativ, weil dort die Beobachtungen mit den teureren Häusern fehlen würden. Um diesem Problem entgegenzuwirken, wird die Reihenfolge der Beobachtungen in der Datenmatrix durch das Mischen vor der Aufteilung verändert. Wenn du das aus irgendeinem Grund nicht wünschst, musst du den Parameter *shuffle* auf *False* setzen.

Ein weiterer Parameter *random_state* steuert das Mischen der Daten vor der Aufteilung. Wenn diesem Parameter eine ganze Zahl zugewiesen wird, werden die Beobachtungen in der Datenmatrix über mehrere Funktionsaufrufe hinweg auf die gleiche Weise gemischt. Das gewährleistet die Reproduzierbarkeit der Ergebnisse. Wenn man diesem Parameter keinen Wert zuweist, werden die Beobachtungen bei jedem Aufruf der Funktion auf unterschiedlichen Weisen gemischt. Damit du beim Ausprobieren dieses Notebooks die gleiche Zerlegung des Datensatzes wie im Video erhältst, weisen wir dem Parameter *random_state* einen Wert zu, zum Beispiel 45.

```
# Zerlege den Datensatz in die Trainings- und Testmenge
X_train, X_test, y_train, y_test =\
    model_selection.train_test_split(X_california_housing,
                                     y_california_housing,
```

```
                                        test_size = 0.1,
                                        random_state = 45)
```

Lineares Regressionsmodell

Ein lineares Regressionsmodell wird mit der Funktion *LinearRegression()* des *Scikit-learn*-Untermoduls *Linear Models* erstellt. Mit der Funktion *fit()* wird das Modell trainiert bzw. an die gegebenen Daten angepasst. Diese Funktion erhält die Trainingsmenge und das entsprechende Zielarray als Eingabe.

Quelle [6]

```
# Erstelle ein lineares Regressionsmodell
lin_reg = linear_model.LinearRegression()

# Trainiere das Modell mit der Trainingsmenge
lin_reg.fit(X_train, y_train)
```

Jetzt haben wir ein fertig trainiertes Regressionsmodell, mit dem wir den mittleren Hauswert für alle Beobachtungen unserer Testmenge schätzen können. Das erfolgt mit der Funktion *predict()*.

```
# Erstelle die Vorhersage für die gesamte Testmenge
y_pred = lin_reg.predict(X_test)
```

Um sehen zu können, wie gut unser Regressionsmodell die Werte des Zielmerkmals schätzen kann, geben wir exemplarisch für die erste Beobachtung der Testmenge das mittlere Einkommen, den tatsächlichen und den durch unser Modell geschätzten mittleren Hauswert aus.

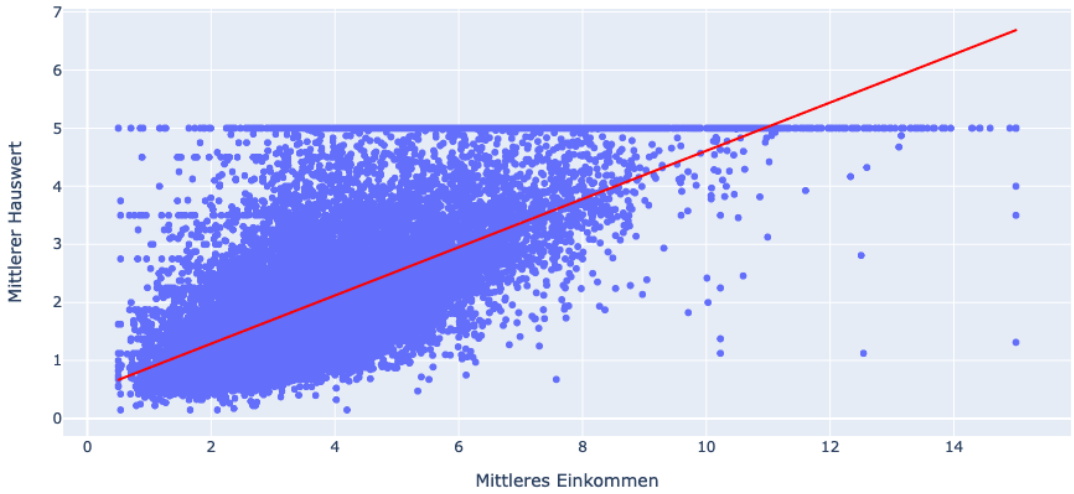
```
# Gebe den Wert des beschreibenden Merkmals für
# die erste Beobachtung der Testmenge aus
print("Das mittlere Einkommen der ersten "
      "Beobachtung aus der Testmenge:", X_test[0,0])

# Gebe den tatsächlichen Wert des Zielmerkmals für
# die erste Beobachtung der Testmenge aus
print("Der tatsächliche mittlere Hauswert der "
      "ersten Beobachtung aus der Testmenge:", y_test[0])

# Gebe den geschätzten Wert des Zielmerkmals für
# die erste Beobachtung der Testmenge aus
print("Der geschätzte mittlere Hauswert der "
      "ersten Beobachtung aus der Testmenge:", y_pred[0])
```

In der Ausgabe kannst du sehen, dass der geschätzte Wert (1,46) von dem tatsächlichen Wert (0,66) etwas (oder doch lieber stark) abweicht. Das liegt daran, dass bei der linearen Regression alle Beobachtungen mit geschätzten Werten auf der Regressionsgerade liegen und dadurch natürlich nicht perfekt vorausgesagt werden. Mithilfe der geschätzten Werte können wir die Regressionsgerade zusammen mit den Beobachtungen der Trainingsmenge in einem Bild visualisieren.

```
fig2 = px.line(x=X_test[:,0], y=y_pred, color_discrete_sequence=['red'])
fig3 = go.Figure(data=fig1.data + fig2.data)
fig3.update_layout(xaxis_title='Mittleres Einkommen',
                   yaxis_title='Mittlerer Hauswert')
fig3.show()
```



Take-Home Message

In diesem Video hast du das Python-Modul *Scikit-learn* kennengelernt. Du hast gelernt, wie man Referenzdatensätze aus diesem laden und sie in eine Trainings- und eine Testmenge aufteilen kann. Du kannst jetzt auch ein lineares Regressionsmodell erstellen und es für einen gegebenen Datensatz trainieren. Außerdem kannst du mit Hilfe dieses Modells Werte für das Zielmerkmal für neue Beobachtungen schätzen.

Quellen

Quelle [1] Pedregosa, F *et al.* (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12, 2825-2830.

Quelle [2] Kelley, P.R, Barry, R. (1997) Sparse Spatial Autoregressions. *Statistics and Probability Letters*, 33, 291-297.

Quelle [3] Scikit-learn (2022). Datasets. In *Scikit-learn Reference, Release 1.2.1* https://scikit-learn.org/stable/modules/generated/sklearn.datasets.fetch_california_housing.html

Quelle [4] Scikit-learn (2022). California Housing dataset. In *Scikit-learn Reference, Release 1.2.1* https://scikit-learn.org/stable/datasets/real_world.html#california-housing-dataset

Quelle [5] Scikit-learn (2022). Model Selection. In *Scikit-learn Reference, Release 1.2.1* https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

Quelle [6] Scikit-learn (2022). Linear Models. In *Scikit-learn Reference, Release 1.2.1* https://scikit-learn.org/stable/modules/classes.html#module-sklearn.linear_model

Weiterführendes Material

Nguyen, C. N., & Zeigermann, O. (2021). *Machine Learning -- kurz & gut: Eine Einführung mit Python, Pandas und Scikit-Learn*. O'Reilly

Disclaimer